EXTERNAL AUDITORY REPRESENTATIONS OF PROGRAMS: PAST, PRESENT, AND FUTURE–AN AESTHETIC PERSPECTIVE

Paul Vickers

Northumbria University, School of Informatics
Pandon Building, Camden St.
Newcastle upon Tyne, NE2 1XE, UK
paul.vickers@northumbria.ac.uk

ABSTRACT

This paper provides a summary of previous work done in the area of external auditory representations of programs (known as *program auralisation*). A brief historical review is given followed by a short summary of the characteristics of the main program auralisation systems that have been reported in the literature. As program auralisation systems tend to use musical representations they are necessarily affected by artistic considerations. The influence of art theory and practice on the design of computer systems and artefacts (known as *aesthetic computing* [1, 2]) has grown to the extent that there is now a growing field devoted to its study. Therefore, it is instructive to explore the design of program auralisation systems in the light of aesthetic computing ideas. The remainder of this paper then, discusses the main principles of aesthetic computing in relation to program auralisation, and finishes with some views on how aesthetic computing can influence the future development of program auralisation systems.

1. INTRODUCTION

Since Bly's original thesis [3] on the use of sound in interfaces, researchers have been keen to propose that sound be used in software visualisation [4-8]. In software visualisation, mappings are made between software attributes and visual representations that are external to the software (such as graphs, spectra etc.). Such mappings provide a framework for users to construct mental images of the states and structures of software. This works extremely well for components whose features map naturally onto spatial media (such as tree diagrams for genealogy reports or dynamic data structures). Having constructed the mapping between, say, a linked-list data structure and a tree diagram, the programmer forever has a visual metaphor of the data structure which can be abstracted and used to form mental images. When asked to visualise a node being added to the list (Figure 1), the programmer can form an image in their head and so draw a picture showing the new state of the data structure. This visual representation has no direct connection with the actual data structure, it is merely an apt analogue or metaphor.

The aspects of software in which the programmer is interested tend not to be real world objects and thus have no real world auditory signatures. How then can sound be successfully employed in software visualisation?

Computer programming poses an interesting problem for information display. Program events are in the time domain whilst visual mappings are largely spatial representations. Visual techniques give good descriptions of spatial relations and structural details (just like Fourier analysis does for sound waves), but do not naturally represent temporal details. Sound presents us with a complementary modality that increases the diagnostic tools available by giving a temporal view of software (as the wave-form plot does for a sound wave).

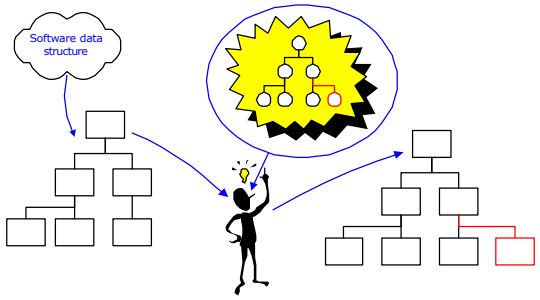


Figure 1 *Metaphor in visualisation. Using the tree metaphor for data structures aids understanding.*

2. HISTORICAL ROOTS

The case for using sound to aid programming was supported by Jackson and Francioni [9], although they felt that a visual presentation was also needed to provide a context or framework for the audio sound track. They argued that some types of programming error (such as those that can be spotted through pattern recognition) are more intuitively obvious to our ears than our eyes. Also, they pointed out that, unlike images, sound can be processed by the brain passively, that is, we can be aware of sounds without needing to listen to them. Francioni and Rover [10] found that sound allows a user to detect patterns of program behaviour and also to detect anomalies with respect to expected patterns.

The use of external auditory representations of program information is known as program auralisation [11]. One of the first attempts at program was described by Sonnenwald et al [8] and was followed by DiGiano [6], DiGiano and Baecker [12], Brown and Hershberger [5] , Jameson [13, 14], Bock [15-17], Mathur et al [18, 19], and Vickers and Alty [20-23]. These early systems all used complex tones in their auditory mappings but, like much other auditory display work, this was done without regard to the musicality of the representations (with the exception of Vickers and Alty). That is, simple mappings were

often employed, such as quantising the value of a data item to a chromatic pitch in the 128-tone range offered by MIDI-compatible tone generators. Furthermore, the pitches were typically atonal in their organisation and were combined with sound effects (e.g. a machine sound to represent a function processing some data). Effort was largely invested in demonstrating that data could be mapped to sound with much less attention given to the aesthetics and usability of the auditory displays. Where aesthetic considerations are taken into account auditory displays become much easier to comprehend. Mayer-Kress et al [24] mapped chaotic attractor functions to musical structures in which the functions' similar but never-the-same regions could be clearly heard. The resultant music could be appreciated in its own right without needing to know how it was produced. Quinn's Seismic Sonata [25] likewise uses the aesthetics of musical form to sonify data from the 1994 Northridge, California earthquake. Alty [26] demonstrated how musical forms could be used to auralise the bubble-sort algorithm.

When we turn from pure data sonification and look towards sonification and auralisation techniques as a complement to existing visualisation models we find that good progress has been made. Brown and Hershberger [5] coupled visual displays of a program during execution with a form of auralisation. They suggested that sound will be a "powerful technique for communicating information about algorithms".

Brown and Hershberger offered some examples of successful uses of audio, for instance, applying sound to the bubble-sort algorithm. The main use of sound in this work was to reinforce visual displays, convey patterns and signal error conditions and was by no means the main focus of the work. Like most other visualisation systems that employ audio, Brown and Hershberger's work used sound as a complement to visual representations. No formal evaluation of the approach was published.

Mayer-Kress et al [24] applied sonification techniques to chaotic attractor functions which led to musical forms in which the similar but never-the-same regions could be heard in a musical framework that is not unpleasant.

Early efforts at pure auralisation were concerned with specific algorithms, often in the parallel programming domain. Examples include Francioni et al [27, 28] who were interested in using auralisations to help debug distributed-memory parallel programs, and Jackson and Francioni [7, 9] who suggested features of parallel programs that would map well to sound. Those systems that are applied to more general sequential programming problems require a degree of expert knowledge to use, whether in terms of programming skill, musical knowledge, expertise in the use of sound generating hardware, or all three.

3. SYSTEMS FOR EXTERNAL AUDITORY REPRESENTATIONS OF PROGRAMS

To-date the following program auralisation tools have been identified (as opposed to visualisation systems that incorporate some sonification): InfoSound [8] by Sonnenwald et al, the LogoMedia system by DiGiano and Baecker [12], Jameson's Sonnet system [13, 14], Bock's Auditory Domain Specification Language (ADSL) [15, 17], the LISTEN Specification Language, or LSL [18, 19, 29], and Vickers and Alty's CAITLIN system [20-23]. All these systems convert their auralisations into MIDI data which are sent, via a MIDI port, to a suitable MIDI-equipped sound generator (such as a music synthesiser, or even a sampler).

3.1. Infosound

InfoSound [8] is an audio-interface tool kit that allows application developers to design and develop audio interfaces. It provides the facility to design musical sequences and everyday sounds, to store designed sounds and to associate sounds with application events. InfoSound combines musical sequences with sound effects. A limitation is that the software developer is expected to compose the musical sequences himself. To the musically untrained (the majority) this militates against its general use.

The system is used by application programs to indicate when an event has occurred during execution and was successfully used to locate errors in a program that was previously deemed to be correct. The authors of InfoSound believe that sound can be a useful feedback mechanism to assist in the debugging process. Users of the system were able to detect rapid, multiple event sequences that are hard to detect visually using text and graphics.

3.2. LogoMedia

LogoMedia [12], an extension to LOGOmotion [4] allows audio to be associated with program events. The programmer annotates the code with probes to track control- and data-flow. As execution of the program causes variables and machine state to change over time the changes can be mapped to sounds to allow execution to be listened to. Like InfoSound, LogoMedia employs both music and sound effects. The authors assert that comprehending "the course of execution of a program and how its data changes is essential to understanding why a program does or does not work. Auralisation expands the possible approaches to elucidating program behaviour" [12].

The main limitation is that the auralisations have to be defined by the programmer for each expression that is required to be monitored during execution. In other words, after entering an expression, the programmer is prompted as to the desired mapping for that expression.

3.3. Sonnet

The Sonnet system [13, 14] is specifically aimed at the debugging process. Using a visual programming language (SVPL) the code to be debugged is tagged with auralisation agents that define how specific sections of code will sound. Figure 2 shows a component to turn a note on and off connected to some source code. The "P" button on the component allows static properties such as pitch and amplitude to be altered. The component has two connections, one to turn on a note (via a MIDI note-on instruction) and one to silence the note (MIDI note-off). In this example the note-on connector is attached to the program just before the loop, and the note-off connector just after the close of the loop. Therefore, this auralisation will cause the note to sound continuously for the duration of the loop. Thus, placement of the connectors defines the auralisation.

Other components allow the user to specify how many iterations of a loop to play. Program data could also be monitored by components that could be attached to identifiers within the code.

The guiding principle is that the programmer, knowing what sounds have been associated with what parts of the program, can listen to the execution looking out for patterns that deviate from the expected path. When a deviation is heard, and assuming the expectation is not in error, the point at which the deviation occurred will indicate where in the program code to look for a bug.

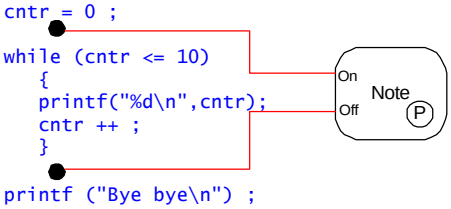


Figure 2 *A Sonnet VPL Component taken from [13]. The VPL component is the box on the right.*

Sonnet is an audio-enhanced debugger. This means that because it interfaces directly with the executing program it is not invasive and does not need to carry out any pre-processing to add the auralisations.

The visual programming language offers great flexibility to the programmer who wants to auralise his program. However, it does require a lot of work if an entire program is to be auralised even very simply.

3.4. ADSL

Bock's Auditory Domain Specification Language (ADSL) [15, 17] differs from the above three approaches in that it does not require sounds to be associated with specific lines of program code. Instead users define *tracks* using the ADSL meta-language to associate audio cues with program constructs and data. These tracks (see Figure 3) are then interpreted by a pre-processor so that the code has the auralisations added to it at compilation allowing the program to be listened to as it runs. The fragment of ADSL in Figure 3 specifies that `for` and `while` loops are to be signalled by playing the 'for_sound' and 'while_sound' respectively. These two sounds have been previously defined and could be a MIDI note sequence, an auditory icon or recorded speech. The sounds will be heard when the keywords for and while are encountered in the program. An advantage of this approach is that it is possible to define a general purpose auralisation. That is, by specifying types of program construct to be auralised there is no requirement to tag individual lines of code with auralisation specifications. When such tagging is required this can be done using the features of the specification language. Like Sonnet, ADSL is non-invasive as the auralisations are added to the source program during a pre-processing phase.

ADSL used a mixture of digitised recordings, synthesised speech and MIDI messages. The choice of sounds and mappings is at the discretion of the user, although a common reusable set of auralisations could be created and shared amongst several programmers. Tracks could be refined to allow probing of specific data items or selective auralisation of loop iterations. The system is flexible, allowing reasonably straightforward whole-program auralisation, or more refined probing (such as in Sonnet).

```
Track_name=Loop
{
1 Track=Status('for'):Snd("for_sound");
2 Track=Status('while'):Snd("while_sound");
}
```

Figure 3 *An ADSL track to monitor for and while loops taken from [15].*

In an experiment [17] thirty post-graduate engineering students from Syracuse University all with some programming experience were required to locate a variety of bugs in three programs using only a pseudo-code representation of the program and the ADSL auditory output. On average, students identified 68% of the bugs in the three programs. However, no control group was used, so it is not possible to determine whether the auralisations assisted in the bug identification.

3.5. Listen

Mathur's *Listen* project [18, 19] follows a similar approach to that used by ADSL. A program is auralised by writing an auralisation specification (ASPEC) in the Listen Specification Language (LSL). A pre-processing phase is used to parse and amend the source program prior to compilation. Again, the original source program is left unchanged.

An ASPEC defines the mapping between program-domain events and auditory events, and an example for an automobile controller is shown in Figure 4. This example auralises all calls to the program functions gear_change, oil_check and weak_battery. The ASPEC contains the auralisation definitions and their usage instructions. For instance, in Figure 4 we see that a call to program function gear_change causes the auralisation gear_change_pattern to be played. This auralisation is defined earlier in the ASPEC as a sequence of MIDI note-on/off instructions.

```
begin auralspec
specmodule call_auralize
var
   gear_change_pattern, oil_check_pattern,
battery_weak_pattern: pattern;
begin call_auralize
   gear_change_pattern:="F2G2F2G2F2G2C1:qq" +
"C1:f";
   oil_check_pattern:="F6G6:h";
   battery_weak_pattern:="A2C2A2C2";
   notify all rule = function_call
"gear_change" using gear_change_pattern;
   notify all rule = function_call "oil_check"
using oil_check_pattern;
   notify all rule = function_call
"battery_weak" using battery_weak_pattern;
end call_auralize;
end auralspec.
```

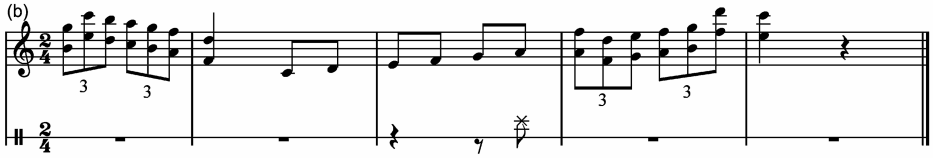
Figure 4 *An LSL ASPEC for an automobile controller. taken from Mathur .*

(a)

```
FOR counter := 1 TO 6 DO
   counter := counter + 1 ;
```

The Pascal code (a) results in the auralization (b). Bar 1 and beat 1 of bar 2 is the tune denoting entry to the loop. The notes remainder of bar 2 and bar 3 representsix iterations of the loop, the final iteration being supplemented by a sleighbell sound. Bars 4 and 5 denote exit from the loop.

(b)



(c)

```
a : = 10 ;
CASE a OF
   '1' : Writeln ('Found 1') ;
   '2' : Writeln ('Found 2') ;
   '3' : Writeln ('Found 3') ;
   '4' : Writeln ('Found 4') ;
   ELSE  Writeln ('Not found)
END ;
```

The Pascal code (c) results in the auralization (d). Bar 1 and beat 1 of bar 2 is the tune denoting entry to the CASE construct. A cowbell sound is played as each case instance is tested (bars 2, 3, and 4). The tune in bars4 to 6 signify exit from the construct. In this example, no match was found for the case selector, so the construct exits in a minor key. In the CAITLIN system the major mode was used to denote Boolean true and minor for Boolean false.

(d)

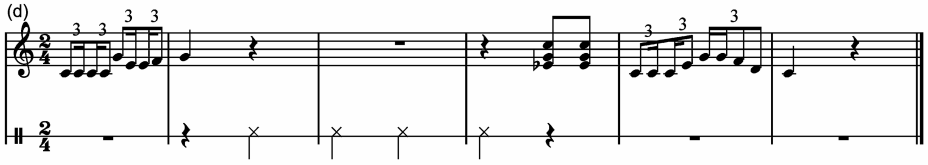


Figure 5 *Example CAITLIN auralisations*

As LSL is a meta-language it can, in theory, be used to define auralisations for programs written in any language; the practice requires an extended version of LSL for each target language.

What is immediately apparent is that writing ASPECS in LSL is not a trivial task as it requires the programmer to learn the syntax of LSL. In addition, some musical knowledge is needed to know how to specify which pitches are used in the auralisations. The Listen project was dormant after 1996 but was reactivated in 2002 and applied to Java in the form of JListen [30]. To-date no formal experimentation or evaluation of the original or newer JListen systems has been published.

3.6. CAITLIN

The CAITLIN system is a non-invasive pre-processor that allows the auralisation of programs written in Turbo Pascal [20-23]. The system was implemented on a relatively modest platform comprising a personal computer and sound card with a General-MIDI-compatible instrument set. Musical output is achieved by sending MIDI data to a multi-timbral synthesiser via the MIDI port on a sound card.

The CAITLIN auralisations were designed around a tonal musical framework. Users were not required to design the auralisation content as the system used unique pre-defined motifs (theme tunes) to represent the Pascal language constructs [22]. This meant that no musical knowledge was needed in order to produce musically-consistent output. Figure 5 shows example auralisations for two program fragments. Experiments with the CAITLIN system indicated that musical auralisations do provide information useful in bug-location and detection tasks [20].

As the CAITLIN system was an experimental prototype it has some limitations. First, the auralisations were only applied to the language constructs (selections and interations) which meant that other program features could not be inspected aurally. Secondly, constructs could not be individually marked for auralisation meaning the entire program was auralised. In a real debugging situation programmers would not monitor an entire program but would choose candidate sections for close scrutiny. This whole-program approach meant that even short programs could take a long time to play back.

4. AESTHETIC CONSIDERATIONS

In any work on auditory display the aesthetics of the sonification are important. One of the goals of the CAITLIN project was to increase the usability of auralisations by drawing upon the well-established aesthetics of tonal music forms. The technical quality of other program auralisation systems was not in question but it is clear that the musical aesthetics they used were highly dependent both on the content and structure of the programs to be auralised and the musical skill of the programmer who had to specify the data-to-pitch mappings. Of course, given sufficient training, any auralisation framework should be usable, but that goes against the more recent efforts to improve computing aesthetics, as espoused and championed, for example, by Norman [31]. The underlying principle of aesthetic computing is that art theory and practice should influence the design of computer systems and artefacts [1]. As art theory and practice embody customisation, personalisation,

preference, culture, and emotion [32] there is much for the designer of the next generation of program auralisation systems to consider.

4.1. Customisation. Personalisation, and Preference

Early program auralisation systems generally let the user define the mappings of data to sound. Whilst this allows almost unlimited customisation, personalisation, and preference it requires some audio-design skills on the part of the user. Customisation and personalisation, then, must be balanced by the knowledge and skills required to make use of them (hence the use of fixed auralisations in the CAITLIN system).

User preference is certainly an important factor. A study of surgeons who listened to music while operating showed that their speed and task accuracy were greater when they listened to self-selected music rather than music chosen by the experimenters [33]. In experiments with CAITLIN we noticed a definite preference amongst subjects for motifs with a strong melody. This was especially apparent in the early pilot studies where the motif design was less refined and some constructs had motifs that were much less musical than others. In the most recent version users expressed a preference for the `FOR` motifs which were also the most melodic, that is, the melodic contour was more elaborate than the scale-based motifs of the selection constructs and the harmonically-richer motifs of the `WHILE` and `REPEAT` loops. Contour was judged by subjects as being a useful aide-mémoire for recalling the motifs [20, 22]. Edworthy [34] and Dowling [35] observed that contour becomes even more important when the tonal context is weak or confusing; contour becomes less important in familiar melodies and melodies retained over a period of time.

Auralisation systems that allow the user to define the auditory mappings have the advantage of supporting preference but with the caveat that the aesthetics of the resultant auralisations may render them less useful than they otherwise might be. This problem could be avoided by providing sets of pre-defined auditory mappings from which the user could choose according to preference.

4.2. Emotion

Sound, and music in particular, clearly has an emotional dimension (after all, we talk of mood music). One emotion that auralisation systems are susceptible to induce is annoyance. Gaver and Smith [36] noted that sounds in the interface can be annoying and that what "seems cute and clever at first may grow tiresome after a few exposures". In our experience, tiresome sounds are usually those that have not been designed with listening aesthetics in mind. Designers go to great lengths to ensure that auditory signals and alarms in safety critical environments (such as aircraft cockpits and nuclear power plants) sit well within their auditory ecology, but these rigours are not so well followed in other auditory displays. It is easy to make a display that clashes with, or masks, other events, or that is simply tiring (both emotionally and cognitively) to use. Approximately half the subjects in an experiment using the CAITLIN system found the auralisations to be moderately annoying, the other half suffering almost no annoyance [37]. The ambiguity of this result gives hope that auralisations can be produced that do not trigger a negative emotional response but cautions us that we must pay very careful attention to this aspect. No studies have been published on the emotive aspects of other auralisation systems but we might expect similar results.

4.3. Culture

With the exception of CAITLIN, the auralisation systems above mapped program data to musical pitches to effect the auralisations without regard to the musicality of the output. Conner and Malmin [38] said that we must recognise that a gap in understanding may exist between the communicator and the receiver. For successful communication between the auditory display and the programmer there must be a common medium in order that the gap may be bridged. Meyer [39] observed that meaning and communication "cannot be separated from the cultural context in which they arise. Apart from the social situation there can be neither meaning nor communication." Music and sonic aesthetics are, to an extent, culturally dependent and so the aesthetics of an auditory display have a pivotal role in determining its success. Watkins and Dyson [40] found that music performed in a style familiar to the listener is easier to recognise and understand. If music forms are used they must not rely on styles that are inaccessible to the average person. That is, the aesthetics must be complementary with, or accessible to, those of the listener. Composers organise music according to defined structures, schemes, or sets of 'rules'. Structuring auralisations according to simple syntactical rules offers the hope of music forming the basis for a bridge of the semantic gap between an incorrectly functioning program and the programmer.

Alty [41] commented that just as furniture designers would never create a chair twelve metres high composers must not produce works that are beyond the cognitive processing capabilities of the listener. Likewise, auralisations must be mappable to different musical and cultural idioms so that the user can select a familiar representation. Just as software interfaces undergo internationalisation to take account of cultural differences and social constructs, so auralisations need to be designed with the listener in mind. What is particularly interesting about music is that recall of melody appears to be an innate skill. That is, people do not need to be trained to recognise melodies. Auralisations that use melodies as carriers of information stand a good chance of being understood and retained in the mind of the listener.

The seven-note diatonic scale, though common in western music, is not, however, "*an inevitable consequence of the psychophysics of tone perception*" [42, p. 5]. Indeed, Hemlholtz believed that the development of musical styles was heavily influenced by culture and aesthetics. In the ancient Greek world there was great debate about the relative merits and vices of the different modal schemes that were common right up until the middle ages. Cultural influence is evident in the divergence of the eastern and western musical traditions. The western classical tradition (especially in the 18th century) was driven by a desire to explore harmony whilst eastern music concentrated on rhythm [see 42 p. 6].

The argument that diatonic systems are in some way more natural than atonal systems is belied by the fact that concert repertoires continue to include new music styles. However, as Parncutt [42] observes, most of the atonal systems have not been incorporated into mainstream (or popular) music as they require more information processing by the listener; the

organising principles of 12-tone music in which there is no tonal centre to the music and each degree of the scale has equal weight (for example, as practised by Schoenberg and Stockhausen) are imperceptible even to trained listeners. Alty [41] explained this in terms of the limits of working memory which, according to Miller [43], can handle around seven bits of information concurrently.

In experiments on subjects' ability to recall melodies, Sloboda and Parker [44] found that the most fundamental feature preserved in a recalled melody was its metrical structure. Further, musicians and non-musicians differed significantly only on one measure – the ability to retain the harmonic structure of the original. Therefore, it is unwise to rely on ability to distinguish harmonic structures in auralisations and this does not commend atonal music systems as good vehicles for auralisation.

Of course, there are cultural as well as perceptual factors at work here as the seven-note diatonic tonal scheme is a western, not a world, music form. That said, there is evidence that the scheme shares characteristics with world musics. Melodies from around the world tend to centre on a particular pitch, thus exhibiting a key feature of tonality [42, p.70]. Furthermore, the twelve-note chromatic scale (of which the diatonic scale is a subset) developed independently in different musical cultures (ancient China, India, Persia, and then the west) and the use of the octave, fourth, and fifth intervals (important in tonal forms) is widespread in world musics. Besides, the international success of western rock and pop bands is evidence that even western musical structures are widely accepted across the world, especially in the computer using world [21].

Nevertheless, as designers of auditory displays we must be aware that even a near-universally-accepted idiom (such as western pop music) is not necessarily interpreted the same way around the world for the boundary between sensory and cultural influences is not clear. For example, consonance and dissonance are important concepts but which appear to be specific to western music [42]. This means that understanding, or rather, specific interpretations of particular musical structures cannot be taken for granted. An auralisaton system that uses dissonance to draw attention to exceptional events, for example, may fail for listeners who are more influenced by musical forms that do not place the same emphasis on consonance and dissonance.

5. THE FUTURE?

In an attempt to avoid the pitfalls of requiring programmers to be able to specify good auditory mappings the CAITLIN musical program auralisation system was built with fixed auralisation motifs that were designed within a coherent and self-consistent tonal framework. This helped to ensure that the aural ecology of the system was healthy and that no individual parts of the auralisations dominated the mix or conflicted with others. The benefits of this approach are that the listener receives output that has been designed with aesthetics in mind. The disadvantage is that it is much less configurable to suit different preferences and emotional or cultural needs. In a sense an aural equivalent of XML is needed to allow the content (information or data) to be separated from its presentation (in this case, the auditory metaphor). Then designers could produce sets of auditory mappings in much the same way that visual interfaces (or skins) are produced for popular programs today. For example, there could be a jazz set, a Bach chorale set, a Javanese Gamelan music set, or even a Chinese classical opera style. In addition, we envisage providing multiple motifs within each set so that program objects and events can be tagged by the user with the motif of preference in the knowledge that each motif conforms to the aesthetic qualities of the others. Such a development could be considered to be extending the principles of literate programming [45, 46]. Where literate programming tools of the past concentrated on typography and external visual representations to enhance presentation and comprehension of programs [e.g. 47, 48], the tools of the future can make use of auditory and musical aesthetics to extend the programmer's toolbox and visualisation set.

Of course, more experimentation is needed to explore just how sensitive auralisations are to the cultural and aesthetic background of the listener. So far studies have focused on a western tonal system with western subjects. Given the tendency of world music systems to exhibit forms of tonality it would be surprising if the simple musical forms of auralisation systems were not comprehensible to people from other cultures, but this aspect still needs to be explored.

In the pursuit of aesthetic excellence we must be careful not to tip the balance too far in favour of artistic form. Much current art music would not be appropriate for an auralisation system. The vernacular is popular music, the aesthetics of which are often far removed from the ideals of the music theorists and experimentalists. Lucas [49] showed that the recognition accuracy of an auditory display was increased when users were made aware of the display's musical design principles. Watkins and Dyson [40] demonstrated that melodies that follow the rules of western tonal music are easier to learn, organise (cognitively), and discriminate between than control tone sequences of similar complexity. So, it would seem that the cognitive organisational overhead associated with atonal systems makes them less well suited as carriers of program information.

The sonifications of Quinn [25], King and Angus [50], and Mayer-Kress et al [24] had a dual function to stand as music in their own right but also to shed new light on the underlying data. However, the purpose of auralisation systems is not entertainment or the communication of mood and emotion, but solely to assist programmers with understanding software and its behaviour. The intentional product of an auralisation system is the communication of information or knowledge with music as the carrier. The music itself, inasmuch as it exists as an entity in its own right, is not the intentional product but a by-product of the auralisation process. Therefore, whatever music systems and aesthetics are employed they must not detract from the prime purpose which is to communicate information.

In addition to exploring the cultural aspects of musical auralisation it also remains to see how auditory and visual mappings work together. It is to be hoped that the temporal/spatial communication space provided by an audio-visual auralisation system will provide a powerful set of tools to help programmers with writing, comprehending, and debugging their code. For example, one can envisage an auralisation tool in which a graphical visualisation displays the state of a data structure whilst an auralisation communicated program control flow. The use of a bi-modal system offers exciting opportunities for program comprehension and debugging tasks. The ease with which music and non-speech audio can now be incorporated

into programming environments (especially the Java platform) means that such a system is a realisable goal in the short to medium term. Integrating auralisation tools into the IDE is necessary to allow common debugging techniques such as breakpoints and step/trace facilities to be extended into the auditory domain.

6. CONCLUSIONS

The first generation of program auralisation systems provided proof of concept and technical feasibility. For this research to continue fruitfully, the next set of tools for adding external auditory representations of programs must take account, not just of interaction design principles, but also of the aesthetics of the tool and the aesthetics of the context and culture in which the tool is placed.

Support for object-oriented multi-threaded environments is necessary. The potential for sound (and music in particular) to assist with comprehension in such a context needs to be explored. The orchestral model of families of timbres might be usefully applied to enable programmers to distinguish between the activities of different thread.

7. ADDITIONAL FILES

Audio examples from the CAITLIN system can be heard by visiting www.auralisation.org.

8. REFERENCES

[1] P. Fishwick, "Aesthetic Programming: Crafting Personalized Software," *Leonardo*, vol. 35, pp. 383-390, 2002.

[2] P. Fishwick, "Aesthetic Computing - in preparation," MIT Press, 2004.

[3] S. A. Bly, "Sound and Computer Information Presentation," University of California, Davis, 1982.

[4] R. M. Baecker and J. Buchanan, "A Programmer's Interface: A Visually Enhanced and Animated Programming Environment," presented at Twenty-Third Annual Hawaii International Conference on Systems Sciences, 1990.

[5] M. H. Brown and J. Hershberger, "Color and Sound in Algorithm Animation," *Computer*, vol. 25, pp. 52-63, 1992.

[6] C. J. DiGiano, "Visualizing Program Behaviour Using Non-speech Audio," in *Computer Science*. Toronto: University of Toronto, 1992.

[7] J. A. Jackson and J. M. Francioni, "Synchronization of Visual and Aural Parallel Program Performance Data," in *Auditory Display*, vol. XVIII, *Santa Fe Institute, Studies in the Sciences of Complexity Proceedings*, G. Kramer, Ed. Reading, MA: Addison-Wesley, 1994, pp. 291-306.

[8] D. H. Sonnenwald, B. Gopinath, G. O. Haberman, W. M. Keese, III, and J. S. Myers, "InfoSound: An Audio Aid to Program Comprehension," presented at Twenty-Third Hawaii International Conference on System Sciences, 1990.

[9] J. A. Jackson and J. M. Francioni, "Aural Signatures of Parallel Programs," presented at Twenty-Fifth Hawaii International Conference on System Sciences, 1992.

[10] J. M. Francioni and D. T. Rover, "Visual-Aural Representations of Performance for a Scalable Application Program," presented at High Performance Computing Conference, 1992.

[11] G. Kramer, "Auditory Display." Reading, MA: Addison-Wesley, 1994.

[12] C. J. DiGiano and R. M. Baecker, "Program Auralization: Sound Enhancements to the Programming Environment," presented at Graphics Interface '92, 1992.

[13] D. H. Jameson, "Sonnet: Audio-Enhanced Monitoring and Debugging," in *Auditory Display*, vol. XVIII, *Santa Fe Institute, Studies in the Sciences of Complexity Proceedings*, G. Kramer, Ed. Reading, MA: Addison-Wesley, 1994, pp. 253-265.

[14] D. H. Jameson, "The Run-Time Components of Sonnet," presented at ICAD '94 Second International Conference on Auditory Display, Santa Fe, NM, 1994.

[15] D. S. Bock, "ADSL: An Auditory Domain Specification Language for Program Auralization," presented at ICAD '94 Second International Conference on Auditory Display, Santa Fe, NM, 1994.

[16] D. S. Bock, "Sound Enhanced Visualization: A Design Approach Based on Natural Paradigms," in *Graduate School*. Syracuse: Syracuse University, 1995, pp. 86.

[17] D. S. Bock, "Auditory Software Fault Diagnosis Using a Sound Domain Specification Language," in *Graduate School*. Syracuse: Syracuse University, 1995, pp. 161.

[18] A. P. Mathur, D. B. Boardman, and V. Khandelwal, "LSL: A Specification Language for Program Auralization," presented at ICAD '94 Second International Conference on Auditory Display, Santa Fe, NM, 1994.

[19] D. B. Boardman and A. P. Mathur, "Preliminary Report on Design Rationale, Syntax, and Semantics of LSL: A Specification Language for Program Auralization," Dept. of Computer Sciences, Purdue University, W. Lafayette, IN Sept. 21 1993.

[20] P. Vickers and J. L. Alty, "When Bugs Sing," *Interacting with Computers*, vol. 14, pp. 793-819, 2002.

[21] P. Vickers and J. L. Alty, "Using Music to Communicate Computing Information," *Interacting with Computers*, vol. 14, pp. 435-456, 2002.

[22] P. Vickers and J. L. Alty, "Musical Program Auralisation: A Structured Approach to Motif Design," *Interacting with Computers*, vol. 14, pp. 457-485, 2002.

[23] P. Vickers and J. L. Alty, "Siren Songs and Swan Songs: Debugging with Music," *Communications of the ACM*, vol. 46, pp. 86-92, 2003.

[24] G. Mayer-Kress, R. Bargar, and I. Choi, "Musical Structures in Data from Chaotic Attractors," in *Auditory Display*, vol. XVIII, *Santa Fe Institute,*

Studies in the Sciences of Complexity Proceedings, G. Kramer, Ed. Reading, MA: Addison-Wesley, 1994, pp. 341-368.

[25] M. Quinn, "Seismic sonata: a musical replay of the 1994 Northridge, California earthquake." Lee, NH: Marty Quinn, 2000.

[26] J. L. Alty, "Can We Use Music in Computer-Human Communication?," in *People and Computers X: Proceedings of HCI '95*, M. A. R. Kirby, A. J. Dix, and J. E. Finlay, Eds. Cambridge: Cambridge University Press, 1995, pp. 409-423.

[27] J. M. Francioni, L. Albright, and J. A. Jackson, "Debugging Parallel Programs Using Sound," *SIGPLAN Notices*, vol. 26, pp. 68-75, 1991.

[28] J. M. Francioni, J. A. Jackson, and L. Albright, "The Sounds of Parallel Programs," presented at 6th Distributed Memory Computing Conference, Portland, Oregon, 1991.

[29] D. B. Boardman, G. Greene, V. Khandelwal, and A. P. Mathur, "LISTEN: A Tool to Investigate the Use of Sound for the Analysis of Program Behaviour," presented at 19th International Computer Software and Applications Conference, Dallas, TX, 1995.

[30] A. P. Mathur. (2004). *Project Listen and JListen*. Retrieved 5 February, 2004, from http://www.cs.purdue.edu/homes/apm/listen.html

[31] D. A. Norman, *Emotional Design: Why We Love (or Hate) Everyday Things*: Basic Books, 2004.

[32] P. Fishwick, "Personal Communication," 2003.

[33] K. Allen and J. Blascovich, "Effects of music on cardiovascular reactivity among surgeons," *Journal of the American Medical Association*, vol. 272, pp. 882-884, 1994.

[34] J. Edworthy, "Melodic Contour and Musical Structure," in *Musical Structure and Cognition*, P. Howell, I. Cross, and R. West, Eds. New York: Academic Press, 1985, pp. 169-188.

[35] W. J. Dowling, "Melodic Information Processing and Its Development," in *The Psychology of Music*, D. Deutsch, Ed. New York: Academic Press, 1982, pp. 413-429.

[36] W. W. Gaver and R. B. Smith, "Auditory Icons in Large-Scale Collaborative Environments," presented at Human-Computer Interaction: Interact '90, Cambridge, UK, 1990.

[37] P. Vickers, "CAITLIN: Implementation of a Musical Program Auralisation System to Study the Effects on Debugging Tasks as Performed by Novice Pascal Programmers," in *Computer Science*. Loughborough: Loughborough University, 1999, pp. 234.

[38] K. J. Conner and K. Malmin, *Interpreting the Scriptures: A Textbook on How to Interpret the Bible*. Portland, Oregon: Bible Temple Publications, 1983.

[39] L. B. Meyer, *Emotion and Meaning in Music*. Chicago: Chicago University Press, 1956.

[40] A. J. Watkins and M. C. Dyson, "On the Perceptual Organisation of Tone Sequences and Melodies," in *Musical Structure and Cognition*, P. Howell, I. Cross, and R. West, Eds. New York: Academic Press, 1985, pp. 71-119.

[41] J. L. Alty, "Engineering for the mind: cognitive science and musical composition," *Journal of New Music Research*, vol. 31, pp. 249-255, 2002.

[42] R. Parncutt, *Harmony: A Psychoacoustical Approach*. Berlin: Springer-Verlag, 1989.

[43] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information," *Psychological Review*, vol. 63, pp. 81-96, 1956.

[44] J. A. Sloboda and D. H. H. Parker, "Immediate Recall of Melodies," in *Musical Structure and Cognition*, P. Howell, I. Cross, and R. West, Eds. New York: Academic Press, 1985, pp. 143-167.

[45] D. E. Knuth, "Literate Programming," *Computer Journal*, 1984.

[46] J. P. Pardoe and S. J. Wade, "Knuth With Knobs On - Literate Program Development," in *Automating Systems Development*, D. Benyon and S. Skidmore, Eds.: Plenum, 1988.

[47] P. Vickers, J. P. Pardoe, and S. J. Wade, "Software Assisted Program Design," presented at Computer Assisted Learning of Computing, Manchester Polytechnic, 1991.

[48] P. Vickers, J. P. Pardoe, and S. J. Wade, "The Use of Literate Program Development Tools," presented at Software Supported Programming Instruction Workshop, University of Ulster, 1991.

[49] P. A. Lucas, "An Evaluation of the communicative Ability of Auditory Icons and Earcons," presented at ICAD '94 Second International Conference on Auditory Display, Santa Fe, NM, 1994.

[50] R. D. King and C. Angus, "PM—Protein Music," *CABIOS*, vol. 12, pp. 251-252, 1996.